Software features on demand

Keywords : Programming languages; Software variability; Dynamic adaptation; Context-oriented programming; Software as a feature; Feature modelling; Features on demand.

Kim Mens

Abstract – The rapid evolution of smart mobile devices has triggered a growing need for context-aware software. Since the first appearance of context-aware applications in the early nineties, many applications on smart devices today are exhibiting context-aware features. This trend is likely to increase even further in future generations of software systems. Context-aware technology holds the key for software systems to offer the most appropriate behaviour to their users according to the current context of use.

Motivating example. Consider an advanced car dashboard system that provides various features, such as smart driving assistance for novice drivers. When localisation sensors, e.g. on a smartphone connected to the car, detect the car's global position, the system may react by activating a location-specific module with, for example, country-specific driving assistance for driving on the left when in the UK. If the driving assistant for novice drivers would still be active as well, the country-specific assistant should seamlessly adapt to and integrate with the already active one.

Goal. While traditional software variability modeling approaches like *feature modeling* and *software product lines* are evolving to address the increased dynamicity and context specificity required for this new generation of software systems, new paradigms such as *context-oriented programming* [1-3] have emerged. Although developed independently, since they address similar issues, many similarities exist between these approaches. The purpose of this research is to combine the best of these approaches into a novel approach for managing context-aware software variability, to establish a shift from *Software as a Service* (SaaS) to *Features on Demand* [4], where small-grained software features can be added, removed or modified on demand, depending on the current context of use.

Features on Demand. In this novel programming methodology, as opposed to seeing applications as monolithic artefacts conceived for a single purpose, user, or context of use, we regard software applications as more dynamic "clouds of features" that are composed on the fly from sets of *finer-grained features*. Particular services are not pre-composed in a core system, but rather are *dynamically composed* from a set of available features, or adaptations of existing features, as long as they remain *consistent* with each other [5].

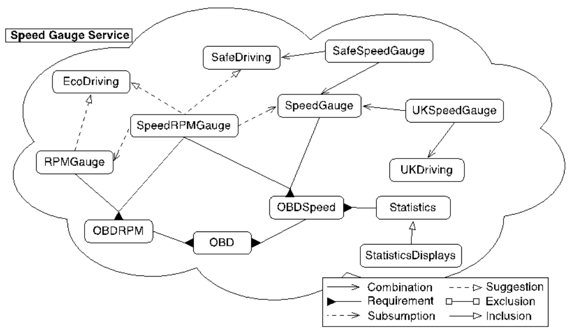

Figure 1: Composing applications from a "cloud of features".

Approach. To achieve our goal, we rely on ideas from three different domains: feature modelling, software as a service, and context-oriented programming. Ideas from feature modelling and software variability management are used as a source of inspiration to *model and represent* features and contexts, as well as their intra- and interdependencies, and how to *verify consistency* of those models. Software as a service is the source of inspiration of our more fine-grained vision of features on demand. Thirdly, context-oriented programming [1-3] is used as implementation technology and to validate our solution, precisely because it supports the key properties needed for conceiving feature clouds: definition of fine-grained features describing new or adapted behaviour, the ability to associate these features with particular contexts of use, declaration of dependencies between features and contexts, and dynamic composition and activation of features depending on the context of use and declared dependencies.

Problems. From a scientific point of view, the four key problems that we investigate are:

1. an appropriate formalism and notation for *representing and modeling* features, the contexts on which they depend, and their intra- and interdependencies;
2. a formalism and semantics [5] for *verifying* both *static and dynamic consistency* of the feature and context models, to ensure a correct behaviour of the composed application;
3. a theory and algorithm for *discovering* the most appropriate *features* for a particular application, user or context of use;
4. a formalism and implementation of a *composition mechanism* allowing the definition of various composition policies and avoiding unexpected interactions between contexts and features upon composition.

Architectural components. Each of these four research problems match a key component in the architecture of our new development environment supporting our *Features on Demand* methodology:

1. a *Feature Store* offering a variety of features from which user applications can be built (or adapted);
2. a *Feature Verifier* which can be used to verify consistency between different features in the store or in an application;
3. a *Feature Discoverer* which gathers relevant information about the context of use and activates or deactivates desired features accordingly;
4. a *Feature Manager* which is aware of the different composition policies and interaction relations between contexts and features, and which takes these into account to guarantee that the system remains coherent and exhibits the desired behaviour.

References

[1] S. Gonzalez, K. Mens et al. "Context-Oriented Programming with the Ambient Object System". Universal Computer Science, 2008.

[2] S. Gonzalez, N. Cardozo, K. Mens et al. "Subjective-C: Bringing Context to Mobile Platform Programming". SLE 2010.

[3] S. Gonzalez, K. Mens et al. "Context Traits: Dynamic behaviour adaptation through run-time trait recomposition". AOSD 2013.

[4] N. Cardozo, K. Mens et al. "Features on Demand". VaMoS 2014.

[5] N. Cardozo, S. Gonzalez et al. "Semantics for Consistent Activation in Context-Oriented Systems". IST 2015.